Knowledge representation

RDF and schemaless databases

Tanel Tammet

TTU

Lecture overview

Multicol tables vs few-and-fixed-col tables

Key-value pairs

RDF high-level overview

Problems and complexities with few-and-fixed-col tables

Namespaces

RDF details

The relational databases have been a standard way to store and query data for decades

Implementations are complex and polished

SQL is everywhere

Alternatives: existing and sought

- Network databases
- Object-relational mapping
- Main memory databases
- Nosql movement
- Document databases
- XML databases
- RDF, Sparql, semantic web
- Google Bigtable and MapReduce framework

Some major powers

Oracle: times ten memory database

Oracle: RDF query extension to SQL

SAP: in-memory database

IBM: research, experiments in sem web

HP: jena rdfs+owl system

Google: BigTable etc

MSFT: profile manager and

reinventing “own”RDF

Key-value pairs

- Extremely common in ordinary programming
- Easy to make deeply nested structures
- Somewhat different from the relational model of SQL (conversions always possible, still ...)
- Specialized databases a better match:
  - Network databases
  - Key/value databases
  - RDF and similar databases

Association lists

- Aka associative array, associative container, map, mapping, hash, dictionary, finite map, and in query-processing an index or index file
- Conceptually

  get_value(array,key) returns value

  set_value(array,key,value) sets value
- Simple standard arrays are a special case:

  array[10] returns value at pos/key 10

  array[10]=101 sets value at pos/key 10

Different languages

- Perl *hashes*: hash-based implementation

```
$age{“John Smith”}=32;
$val{“zero”}=0;
$val{“one”}=1;

%attribs = (
    "ID" => "joe",
    "style" => "color:blue;",
    "onClick" => "check();"
);
```

Different languages

- Javascript *arrays*: hash-based implementation

```
var myCars=new Array();
myCars[1]=“Opel";
myCars[3]=“Ford";
myCars[“recent”]=“Volvo”
```

Different languages

- Scheme *a-lists*: list-based implementation

(1 . 3) pair of two pointers

(1 (2 3) “aa” (“b”)) sht for (1 . ((2 . (3 . ()))) ...

(set alist ‘((0 “Opel”) (“recent” “volvo”)))

(assoc alist “recent”) returns “volvo”

Simple database systems

- Simplest database engines just implement fast associative array storage on a disk. Berkeley db python usage example:

```
import bsddb
db = bsddb.btopen('/tmp/spam.db', 'c')
for i in range(10):
    db['%d'%i] = '%d'% (i*i)

db['3']      gives  '9'
db.keys()  gives  ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
db.first()   gives  ('0', '0')
db.next()  gives  ('1', '1')
db.last()    gives ('9', '81')
```

All database systems

- Database indexes used by all database systems are also associative array storage systems.

Google base

- See http://code.google.com/apis/base/
- And http://www.google.com/base/api/demo/html/demo.html

  - Search:

http://base.google.com/base/feeds/snippets?
maxresults=250&
bq=[author : Shakespeare]

XML

- Structured text markup language
- Pure syntax: tags, attributes have no predefined meaning
- Everybody is allowed to invent their own meaning/usage for tags

XML standard rep example

- Pack information between tags:

```
<beer>
  <name>Nuia tume</name>
  <producer>Nuia õllevabrik</producer>
  <type>Porter</type>
  <cost>8.90</cost>
  <strength>5.4</strength>
<beer/>
```

Tags

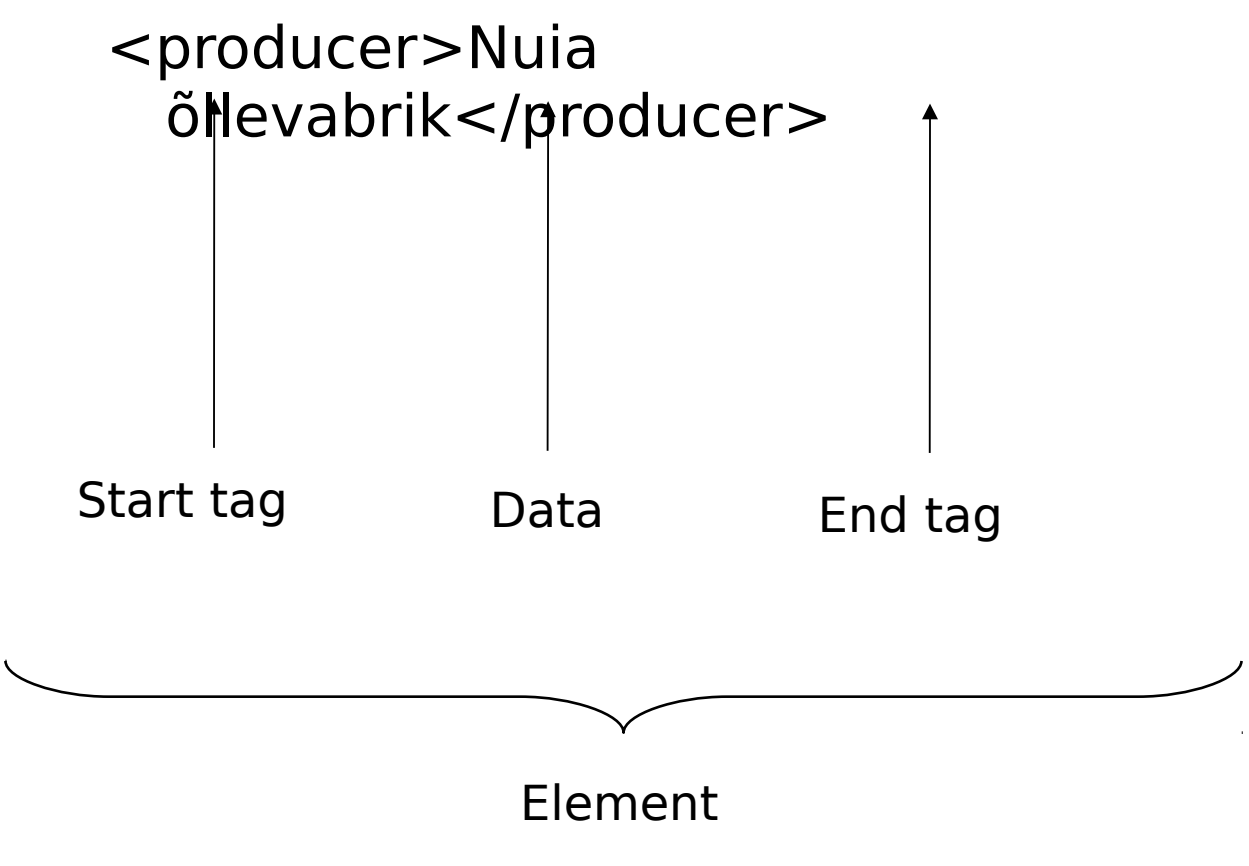
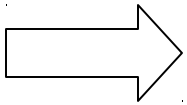
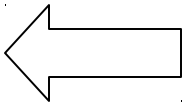
- 
<producer>Nuia
õlevabrik</producer>
Start tag
Data
End tag
Element

Table data example

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3






```
<R>
   <tuple>
      <A>a1</A>
      <B>b1</B>
      <C>c1</C>
   </tuple>
   <tuple>
      <A>a2</A>
      <B>b2</B>
      <C>c2</C>
   </tuple>
   ...
</R>
```

Deep nesting ok

http://mylang.com/1 name http://mylang.com/2
http://mylang.com/2 xx:stringvalue “Nuia dark”
http://mylang.com/2 lang “est”

http://mylang.com/1 name “Nuia dark”

http://mylang.com/1 producer http://mylang.com/3

```
<beer>
 <name lang=“est”>Nuia tume</name>
 <name lang=“eng”>Nuia dark</name>
 <producer>
    <name>Nuia õllevabrik</name>
    <address>
      <city>Nuia</city>
       <street>Karu</street>
    </address>
 </producer>
 <type>Porter</type>
 <cost>8.90</cost>
 <strength>5.4</strength>
<beer/>
```

Nimeruumid (namespaces)

- XML formaadi üks põhieelis on see, et:
  - Kui kahel erineval keelel A ja B on mõlemal XML süntaks, siis
  - saab A keelse teksti sisse panna teksti keeles B, ja tulemus on korrektne XML.

- Mis juhtub aga siis, kui nii keeles A kui B on sama tag, aga eri tähendusega? Siis me ei saa kokkupandud tekstis aru, kumb tähendus võtta.

```
<table>
  <tr>
  <td>Apples</td>
  <td>Bananas</td>
  </tr>
</table>
```

```
<table>
  <name>African
    Coffee Table
  </name>
  <width>80</width>
  <length>120</length>
</table>
```

Nimeruumid

- Lahenduse idee: paneme iga tagi ette kooloniga eraldatult keele nime.
- Võtame vasakul tabelis (HTML) keele nimeks h ja paremal (furniture) f:

```
<h:table>
  <h:tr>
  <h:td>Apples</h:td>
  <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

```
<f:table>
  <f:name>African
    Coffee Table
  </f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

- Nüüd on vasakul ja paremal erinevad tagid, konflikti pole.

Nimeruumid

- **Kuidas aga tagada, et kaks eri keelt ei võtaks sama nime?**
- **Idee:**
  - Olgu keelte nimed (soovitavalt) hästi pikad.
  - Kasutame nimena URL-i (a la http://www.ttu.ee/it/xkeel)
    - Siis saab keele inmloetava kirjelduse panna selle URLi peale.
    - Siis on igal keeletegijal mõttekas kasutada URL-i, kuhu ta saab oma kirjelduse riputada.
    - Näiteks, kui ma teeksin oma keele nimega http://www.microsoft.com/xlang, siis vaevalt mul õnnestub sellele URL-ile mingit teksti riputada!
    - Siis on vähe tõenäoline, et eri keelte nimed kattuvad.
- **Probleem:** pikka keele nime on ebameeldiv iga tagi ette kirjutada: <http://www.ttu.ee/it/xkeel:table> oleks halb.
- **Lahendus:** kasutame tagi alguses nime lühendit, mis on XML tekstis varem defineeritud.

Nimeruumid

- XML nimeruumide standard näeb ette:
  - Iga tagi ette võib olla kirjutatud, mis keeles ta on

<keelelyhend:tag>

  - Nime lisamine tagile ei ole kohustuslik.
  - Keelte nimedeks kasutatakse URI-sid (URL-e)
  - Lühendite definitsioonid on spetsiaalatribuudid tagidel kujuga **xmlns:lyhend=”taisnimi”**

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:ex="http://www.example.org/terms/">
  <rdf:Description rdf:about="http://www.example.org/index.html">
   <ex:creation-date>August 16, 1999</ex:creation-date>
   <ex:language>English</ex:language>
   <dc:creator rdf:resource="http://www.example.org/staffid/85740"/>
  </rdf:Description>
</rdf:RDF>
```

Nimeruumid

- **Nimedefinitsiooni kehtivuspiirkond tekstis on piiratud:**
  - Mingis tagis xmlns abil defineeritud keelenime lühend kehtib nii selle tagi enda jaoks (vt eelmist näidet) kui kõigi tema sees (kuitahes sügaval) olevate tagide ja atribuutide jaoks.

- Mugavuse jaoks näeb standard ette **vaikimisi nimeruumi** defineerimise võimaluse **ilma lühendita**:

```
<table xmlns="http://www.ttu.ee/it/xkeel">
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

- Mispeale tagil ja kõigil sisemistel tagidel on automaatselt see nimeruum, kui neile pole eraldi antud mõnda teist nimeruumi.
- Atribuutidele peab nimeruumi aga siiski andma kooloniga!

Nimeruumid: URI, URL ja URN

- URI (Uniform Resource Identifier) on globaalne, unikaalne nimi, mis võib olla kas URL või URN (mõlemad on URI-d).
- URL (uniform resource locator) on kujul

  yhendusprotokoll:masin/asukohtmasinas

  näiteks**:**
  - http://www.ttu.ee/it/index.html
  - ftp://ftp.is.co.za/rfc/rfc1808.txt
  - mailto:tammet@staff.ttu.ee

- URN (uniform resource name) on unikaalne nimi kujul

  urn:nimi

  näiteks**:**
  - urn:ttu.ee

Association-list style representation

- Lisp syntax with named slots (association list) example:

[‘asas’ ,‘asasa’]
{‘type’: “Porter”, ‘name’: “Nuia tume”}

Item[‘name’]=“Nuia tume”
Item[‘type’]=“Porter”
Data={beer: {‘type’: “Porter”,
‘name’: [{lang: est, str: “Nuia tume”}, {lang: eng, str: “Nuia dark”}],
’producer’: {name: (‘Nuia Õllevabrik)})
}

(beer (text (str “Nuia tume”) (lang „est“))
(name “Nuia dark”)
(producer (name “Nuia õllevabrik”)
(address (city “Nuia”) (street “Karu”))
(type “Porter”)
(cost 8.90)
(strength 5.4))

Schema-less db-s: obvious idea

Each row with N cols is represented as N rows of three columns, called sometimes as

• Row/Object id	Column name	Value
• Object	Property	Value
• Subject	Predicate	Object

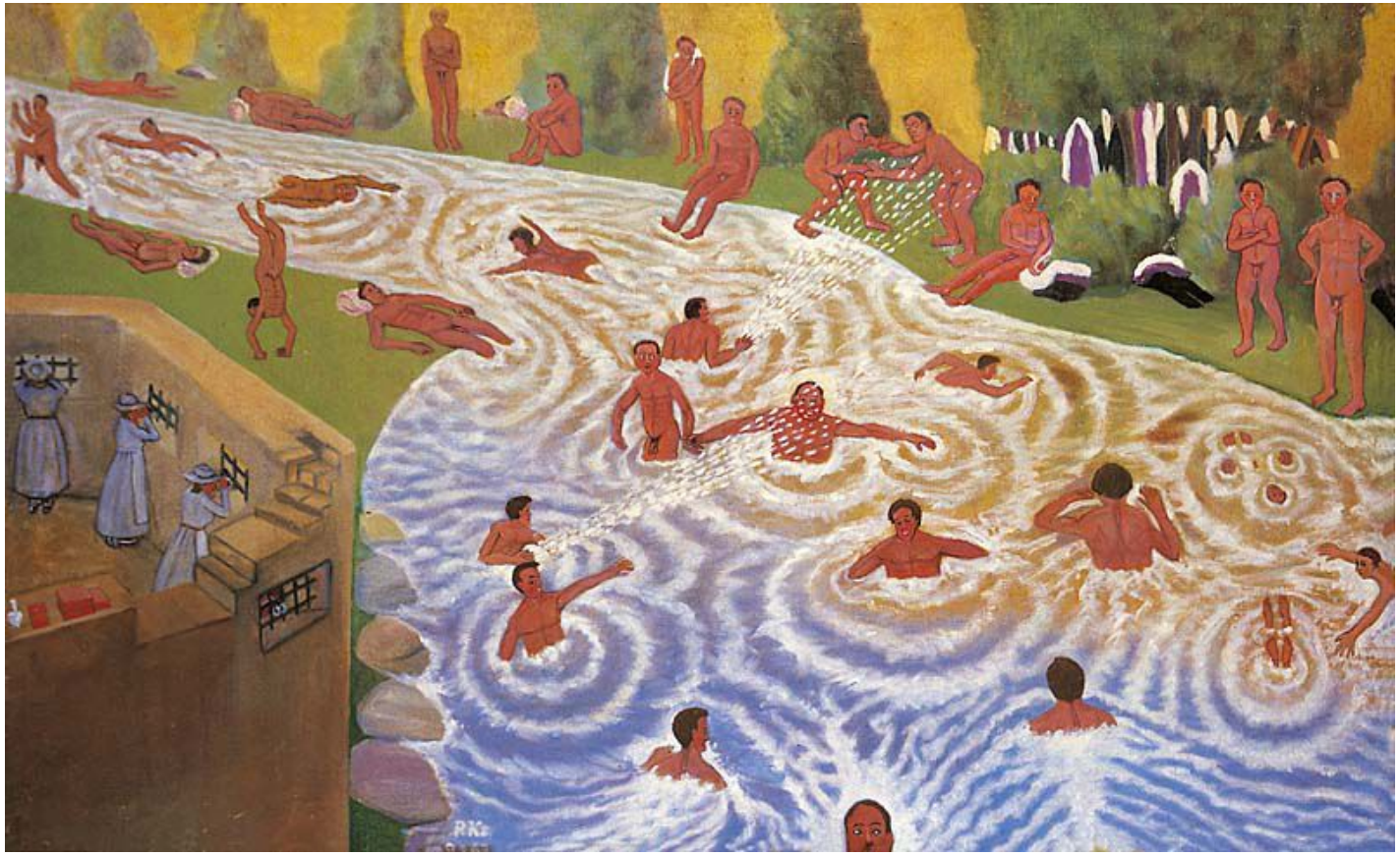
Similar to key-value

Object Property Value

can be combined to

Object:Property Value

Naive?

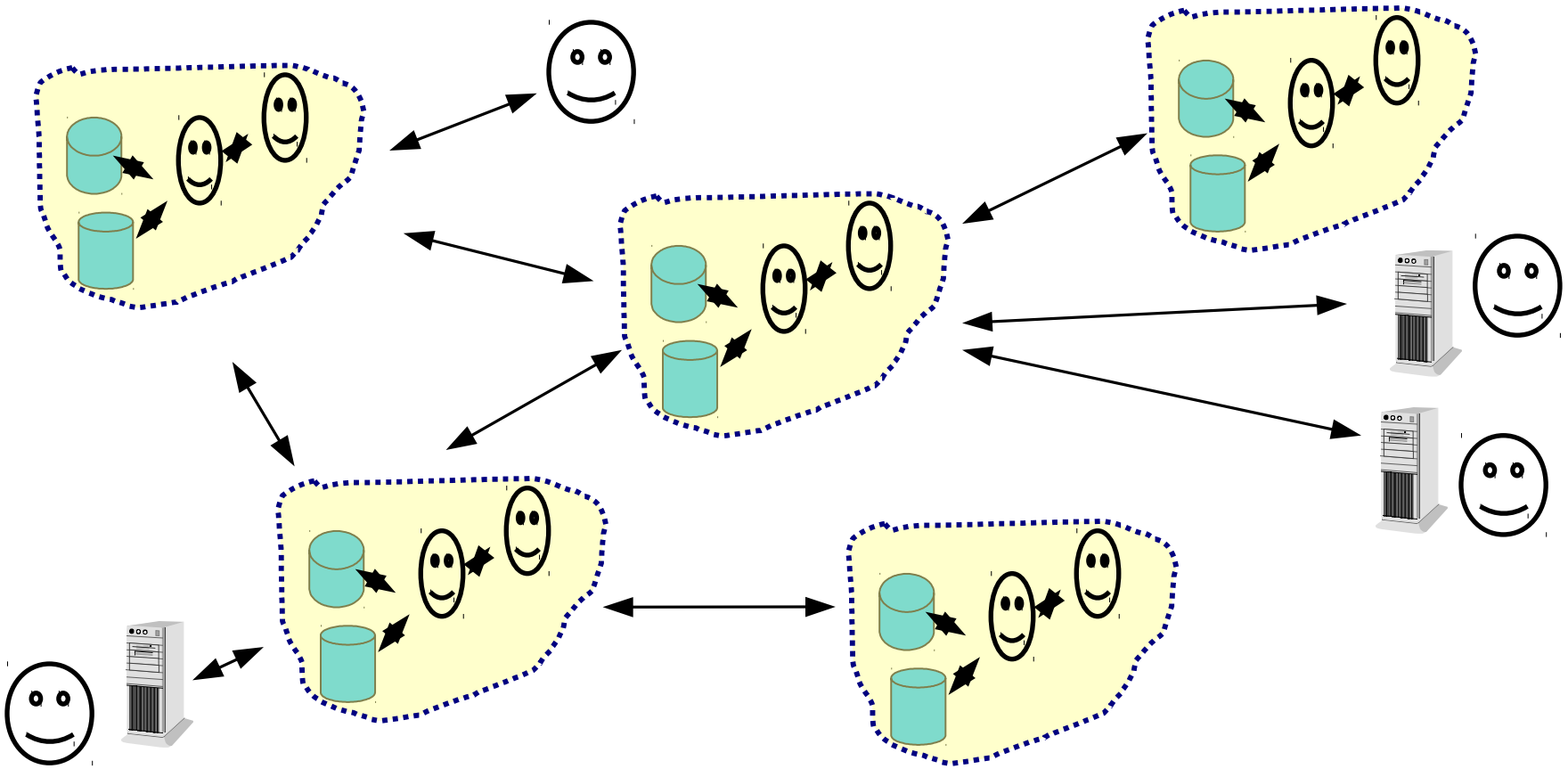


Schema-less is often inevitable

Read data from numerous sources,
aggregate in our own database:

- We have no control over foreign data
- Our understanding of foreign data changes
- Our data sources change

Software is a part of an evolving society



Some case studies from security

CERT, Aarelaid and the global threat data

- Many very different data sources
- Everybody has own format
- Everybody has own meaning of names/values
- Data sources added/changed/deleted
- Conversion has to be done on the fly using rules
- Complex analytical queries are run on the aggregated data

Case study from tourism planner

Sightsplanner project at ELIKO: data scraped, converted, matched, searched, optimal plans built

- Many very different data sources
- Everybody has own format
- Everybody has own meaning of names/values
- Data sources added/changed/deleted
- Before doing processing, conversion has to be done using rules
- Complex queries and search is run on the aggregated data

Back to the RDF(a)

People need schemaless databases. Yet:

- Schemaless databases are slow
- Schemaless databases need rules
- Standards/theory is evolving/changing fast
- Nobody is happy with what we currently have

The main, the only standard, ...

RDF(S): resource description framework

- Developed and pushed by W3C
  http://www.w3.org/TR/rdf-primer/
- Cornerstone of the semantic web project
- Large number of systems supporting
- A lot of tools
- See also wiki

http://en.wikipedia.org/wiki/Resource_Description_Framework

RDF: triple not really a triple

Object Property Value Valuetype

With valuetype normally being either:

- One of xml schema datatypes
- Global id: URI
- Local id

RDF: some restrictions

Object Property Value Valuetype

Object, Property, Valuetype: URI-s

Value: URI or literal value

Many representation syntaxes

- RDF/XML
- RDFa
- N3
- N-triples
- Turtle
- ….

Example in Turtle syntax

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix contact: <http://www.w3.org/2000/10/swap/pim/contact#>.

<http://www.w3.org/People/EM/contact#me>
      rdf:type     contact:Person;
      contact:fullName    "Eric Miller";
      contact:mailbox      <mailto:em@w3.org>;
      contact:personalTitle   "Dr.".
```

Example in XML syntax

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
  <ex:Person
      rdf:about="http://en.wikipedia.org/wiki/Tony_Benn">
          <dc:title>Tony Benn</dc:title>
          <dc:publisher>Wikipedia</dc:publisher>
   </ex:Person>
</rdf:RDF>
```

Striped XML

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/rdf-syntax-ns#"
            xmlns="http://example.com/some-dlg-schema#">
<Person rdf:id="http://my.com/122">
  <name> John </name>
  <livesWith>
       <Person rdf:id="http://my.com/124" >
            <father>
                <Person rdf:id="http://my.com/12222" >
                    <name> Fred </name>
                </Person>
           </father>
        </Person>
  </livesWith>
</Person>
</rdf:RDF>
```

How to add metadata to a row?

Like timestamp, changer, row id, status etc etc?

Horrible answer: reification

The ugly head of reification

We have

personid:12 salary 20000

Want to add timestamp and entering person?

The reification way

From

personid:12	salary	2000	+	timestamp etc

To

datarow:10001	subject	personid:12
datarow:10001	predicate	salary
datarow:10001	object	2000
datarow:10001	timestamp	2009-10-20 13:45
datarow:10001	modifier	personid:345

From the relational db ...

One row, N cols in the relational db

First, get N rows of four cols in RDF

Second, get (N*3)+X rows of four cols after reification

N → 12*N

W3C container blunder

RDF provides a *container vocabulary* consisting of three predefined types (together with some associated predefined properties).

A *container* is a resource that contains things. The contained things are called *members*. The members of a container may be resources (including blank nodes) or literals. RDF defines three types of containers:

- rdf:Bag
- rdf:Seq
- rdf:Alt

Problem:

Containers have no real semantics in RDF.

Container semantics would make calc hard.

Containers are fake, pointless temptations.

Local id-s problem

Different object id-s:

- Global URI-s.
  - These are fine.
- Local “blank nodes”.
  - Their semantics/use in the RDF spec is broken: creates unnecessary problems.

Storage in “common” db

Predicate, subject, valuetype URI-s:

- keep a separate table for unique strings
- use numeric string id-s in
  pred,subject,valuetype

Storage in “common” db

Storing value? Can be int, float, string, URI, ...

Several ways, all bad:

- Encode everything as a string
- Encode everything as a number
- Use several columns for different (main) types

Sparql query language

```
PREFIX type: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>
SELECT ?country_name ?population
WHERE {
   ?country   a   type:LandlockedCountries ;
   rdfs:label   ?country_name ;
   prop:populationEstimate  ?population .
FILTER
   (?population > 15000000 &&
     langMatches(lang(?country_name), "EN")) .
} ORDER BY DESC(?population)
```

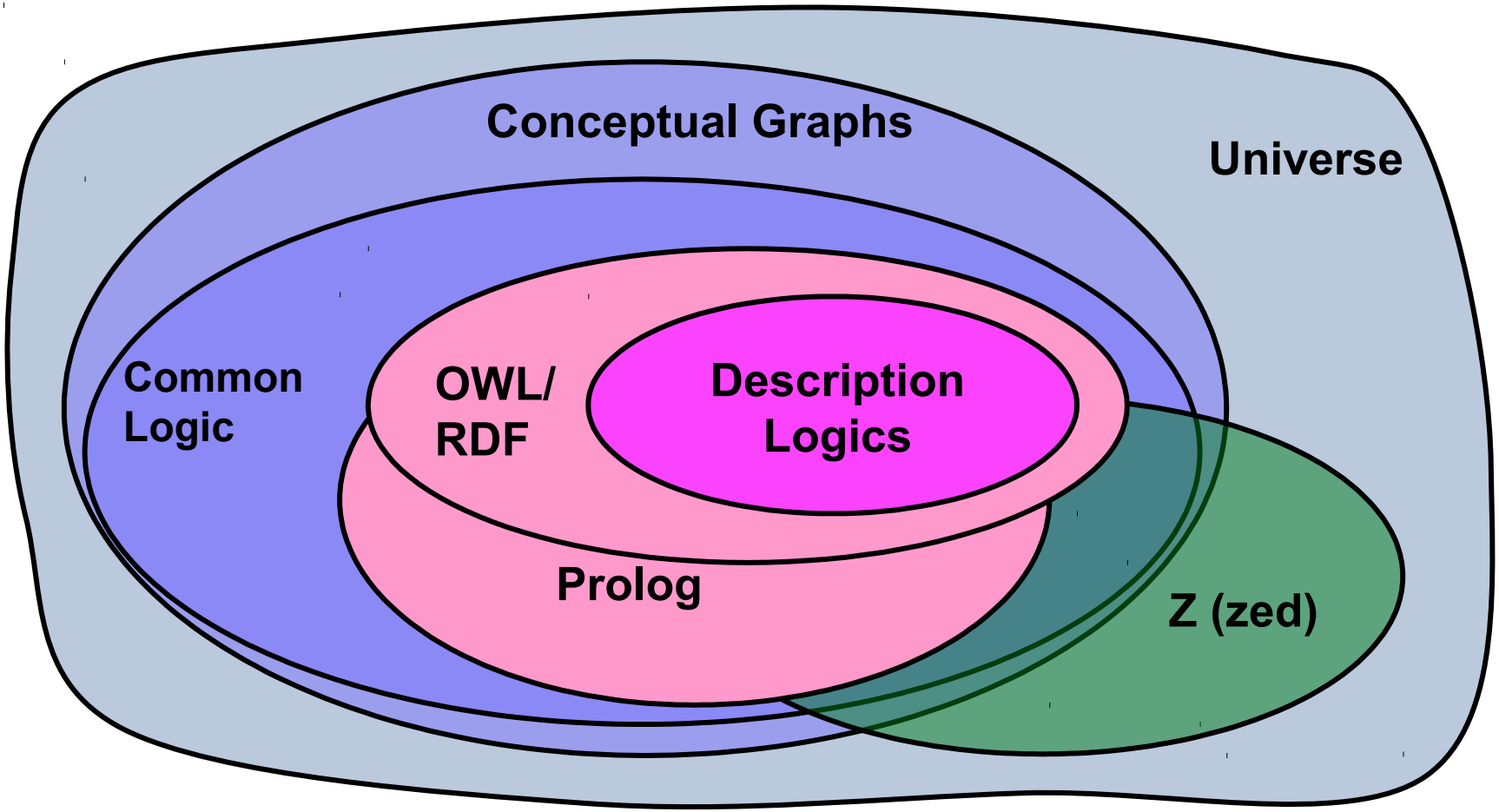
Rule languages for classification

- Simple language: RDFa rules.
  - ex:Van rdfs:subClassOf ex:MotorVehicle .
- Complex language: OWL
  - SameIndividual: f:male g:masculine
  - FunctionalDataProperty: f:hasWife
  - InverseFunctionalDataProperty: f:hasWife
  - Class: f:ReligiousMarriage DisjointWith: f:CivilMarriage
  - Class: f:Marriage EquivalentTo: f:ReligiousMarriage OR f:CivilMarriag

Universal rule languages

- Based on first order logic: sugar and extensions added.
- Each rule engine has own language/features.
- Attempts to create standards:
  - KIF
  - ISO Common logic
  - SWRL
  - RuleML
  - RIF

ISO Common Logic



Several syntaxes, common meaning

$(\forall)(\text{Boy}(x) \rightarrow (\exists)(\text{Girl}(y) \,\&\, \text{Kissed}(x,y)))$

[@every *x]
[If: (Boy ?x) [Then: [*y] (Girl ?y) (Kissed ?x ?y)]]

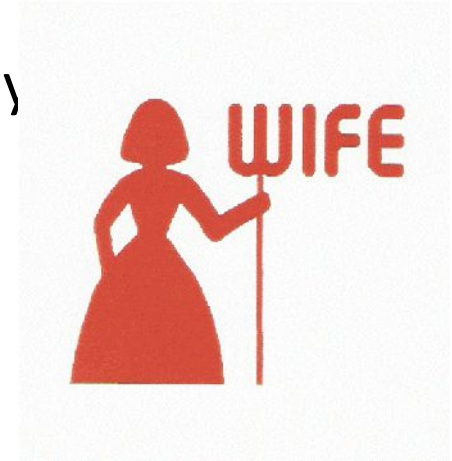
Achieving semantic consistency

System A: (married Jack Jill)

System B: (married (roleset:(husband Jack)(wife Jill)))

A “understand” B? Provide equivalences.

```
(forall (x y)
    (implies  (married x y)
              (married (roleset:(husband x) (wife y
```



W3C RIF: rule interchange format

Again, many syntaxes, common semantics

```
Document (
 Prefix(cpt <http://example.com/concepts#>)
 Prefix(ppl <http://example.com/people#>)
 Prefix(bks <http://example.com/books#>)
 Group (
    Forall ?Buyer ?Item ?Seller (
        cpt:buy(?Buyer ?Item ?Seller) :-  cpt:sell(?Seller ?Item ?Buyer)   )
    cpt:sell(ppl:John bks:LeRif ppl:Mary)   ) )
```

Back to problems

- Most people are not happy with current RDF, OWL or rule languages.
- RDF databases are slow, when compared to relational bases as they are commonly used.
- Rule systems are very slow.

Speed hopes

- RDF is probably better suited for memory db-s
- Locality is less critical in memory than disc:
  - N cols in a row have better locality than N RDF rows of 4 cols
  - Bigger hit for disc based systems than memory systems
- Rule systems are too slow on a disc db: need pure, rule-optimised memory db

Drop triples for N-tuples

Reification is bad.

Put metadata directly into the data row.

Example case:

- ELIKO project: technology for contextual information and user profiles.
- Concrete subproject: intelligent tourist recommendations system

RDF+ schema

Proper RDF data

- Object id (uri reference id)
- Value name (uri reference id)
- Value (encoded in a large numeric)
- Value type (uri reference id)

Metadata on the same row

- Row id (integer)
- Validity start date/time
- Validity end date/time
- Context (uri reference id)
- Trust (integer)
- Timestamp of last modification
- Source id (uri reference id of a web page, database, etc)

Floating suggestions for RDF+

More “ordinary” logic/database system

- N-tuples
- Drop reification
- Drop containers
- Drop restriction that the

  subject and predicate

  must be URI-s
- Drop “blank nodes”
- Simplify value types
- Add negation

See also http://www.slideshare.net/PatHayes/rdf-redux